有限状态自动机的应用

南京外国语学校 贾志鹏

提纲

- 有限状态自动机
- 正则表达式（略）
- Aho-Corasick自动机（重点）
- 后缀自动机（略）

有限状态自动机

- 一个有限状态自动机由下面五部分组成：

1. 一个非空有限状态集合Q
2. 一个输入字母表Σ
3. 一个转移函数δ
4. 一个开始状态$q_0 \in Q$
5. 一个非空接受状态集合$F \subseteq Q$

- 因此常用五元组$(Q, \Sigma, \delta, q_0, F)$来表示一个有限状态自动机$A$。

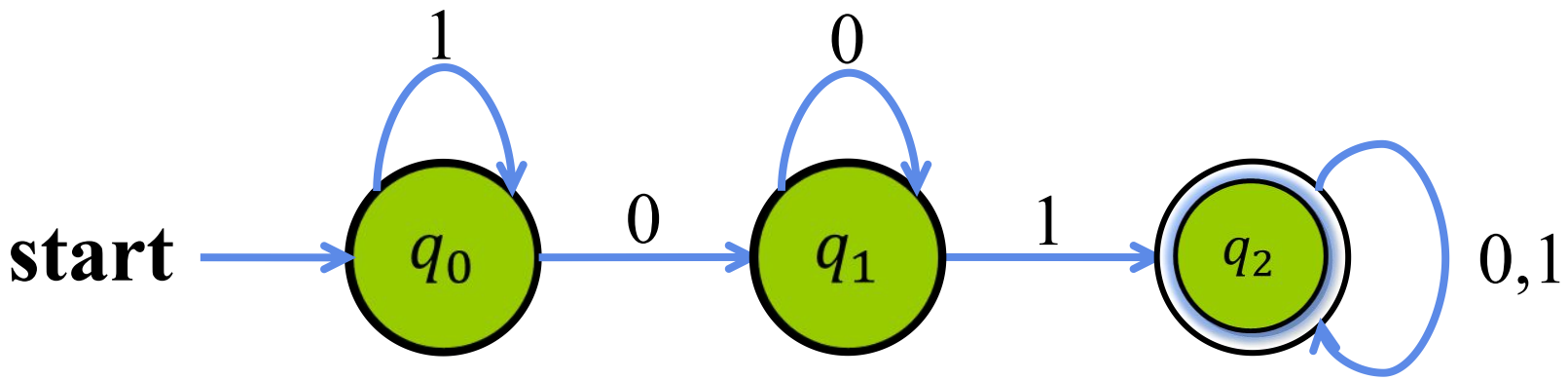
三类有限状态自动机

1. 确定有限状态自动机 (DFA)
2. 非确定有限状态自动机 (NFA)
3. 带ε转移的非确定有限状态自动机 (ε-NFA)

- 它们的差别只在于转移函数δ有所不同。

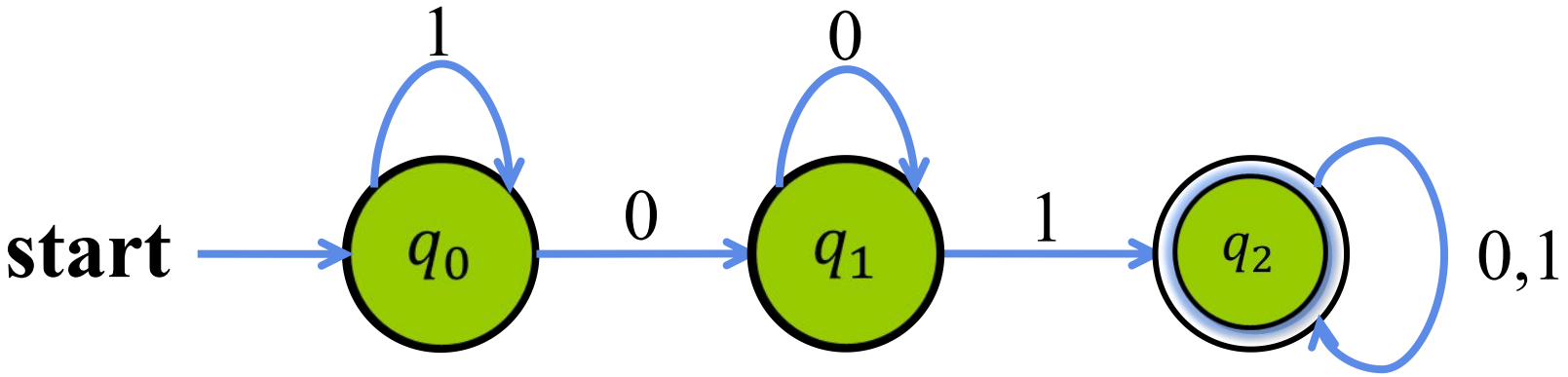
确定有限状态自动机（DFA）

- 转移函数$\delta: Q \times \Sigma \to Q$
- 例如下面的例子（其中字母表$\Sigma = \{0,1\}$、接受状态集合$F = \{q_2\}$）：



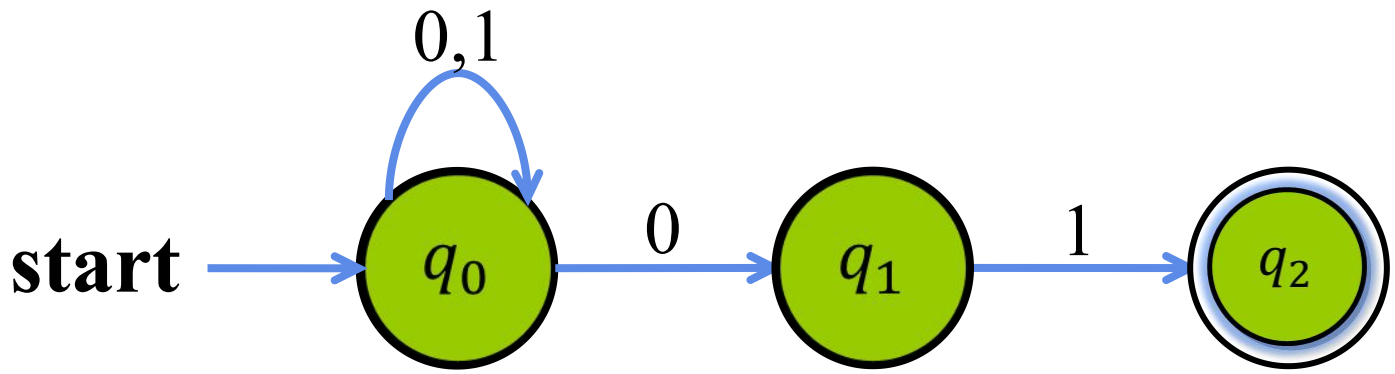
在DFA上识别输入串

- 考虑一个字母表Σ上的串w，从DFA A的开始状态开始走，每次按照w的当前字符选择转移，如果最终走到的状态属于接受状态集合，就说明A接受串w。所有能被A接受的串的集合称为A表示的语言，记做$L(A)$。
- 例如串10010就能被下面的DFA接受。容易看出这个DFA表示的语言是所有包含子串01的串。



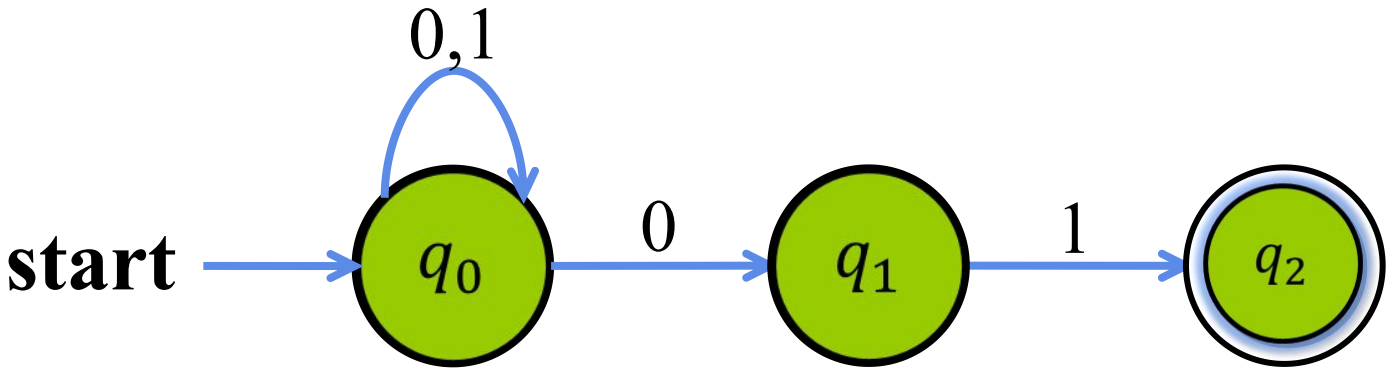
非确定有限状态自动机（NFA）

- NFA与DFA的区别就是，转移函数δ: $Q \times \Sigma \rightarrow P(Q)$，也就是转移到$Q$的某个子集（可以是空集）。例如下面的例子：



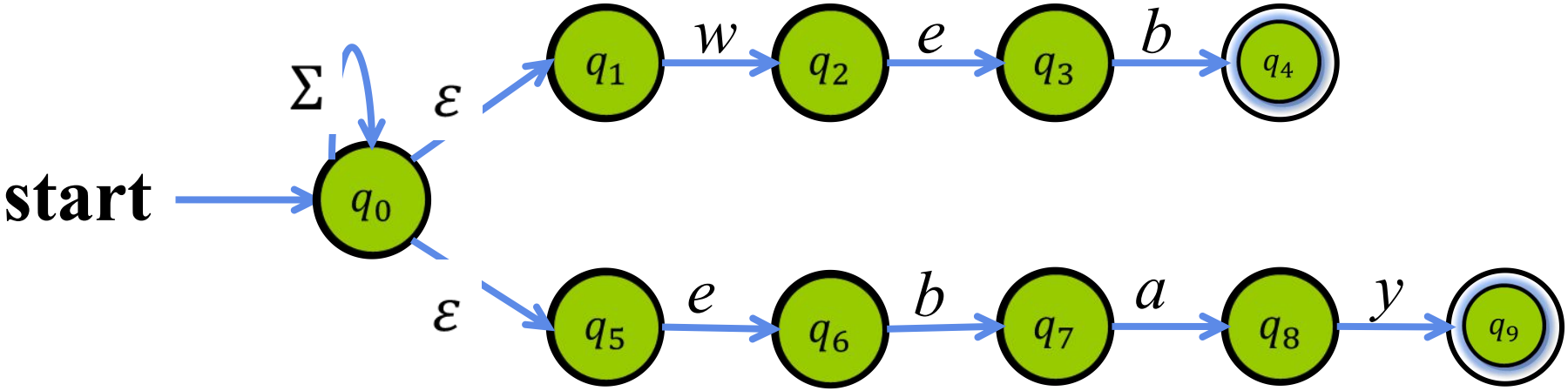
在NFA上识别输入串

- 和在DFA上识别输入串类似，每次根据当前字符往前走。但是由于是转移到状态集合，因此每次都走到的是一个状态集合。如果最终走到的状态集合与接受状态集合的交集不为空，就说明输入串能被接受。同样也能定义一个NFA对应的语言。
- 例如0101能被下面的NFA接受，并且能发现这个NFA的语言就是所有以01结尾的串。



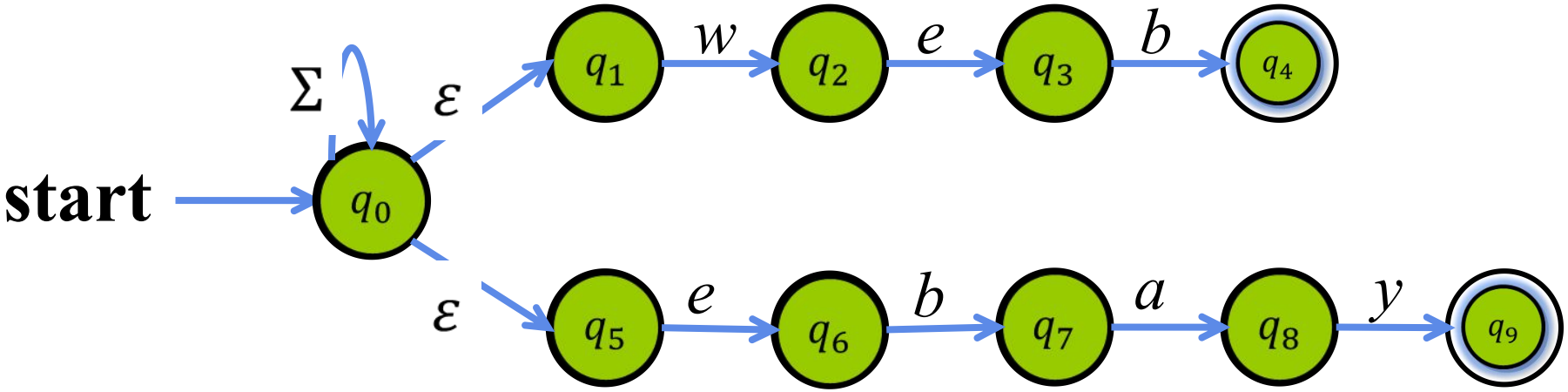
带ε转移的非确定有限状态自动机（ε-NFA）

- ε-NFA与NFA的区别在于，有些转移接受的字符是空字符ε，也就是说可以无条件从一个状态转移到某个状态集合。下面给出一个例子（其中接受状态集合$F = \{q_4, q_9\}$）：



ε-NFA中的ε闭包

- ε-NFA中某个状态q的ε闭包指能从状态q只经过ε转移到达的状态集合。例如q_0的ε闭包是$\{q_0, q_1, q_5\}$。
- 在ε-NFA中识别输入串和在NFA中类似，唯一的不同就是每走到一个状态都要取一次ε闭包（包括刚开始在q_0的时候）。能够看出下面的ε-NFA能够识别所有以*web*或*ebay*结尾的串。

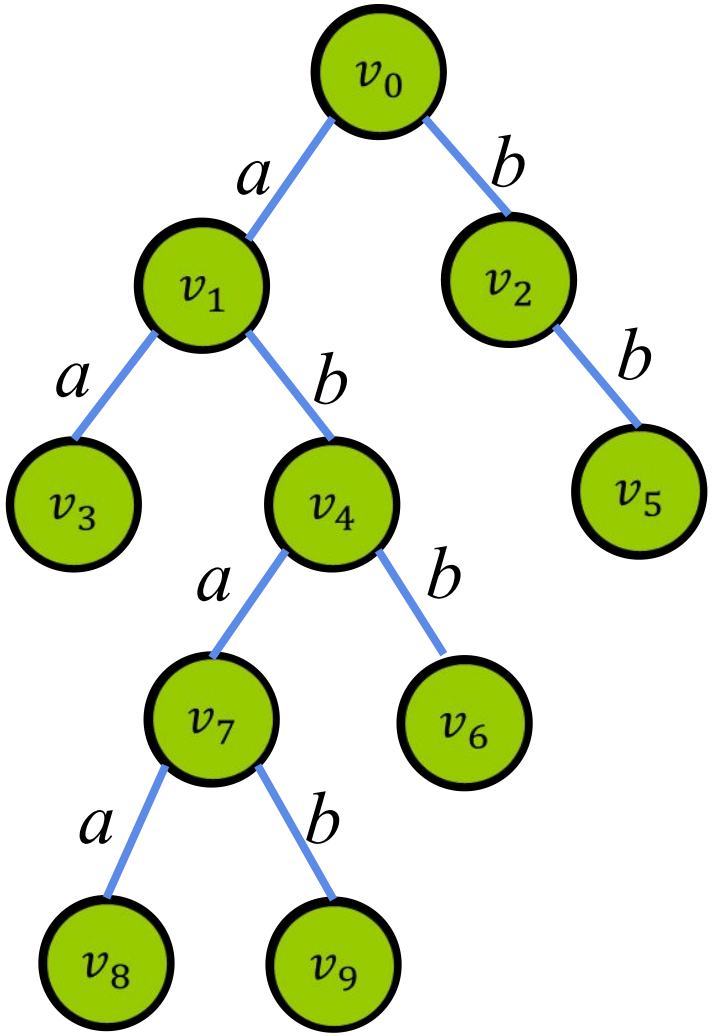


识别输入串的效率

- 假如我们能够$O(1)$获得转移函数，很容易看出，在DFA上识别串w的时间复杂度为$O(|w|)$，而NFA和ε-NFA每次是到达一个状态集合，识别输入串的时间复杂度势必比$O(|w|)$要大，但和具体的输入串与自动机是有关系的。
- 实际上，DFA、NFA和ε-NFA它们能够表示的语言集合是相同的，也就是说对任意一个ε-NFA，都能构造一个表示语言与其相同的DFA。但是对于状态数为m的ε-NFA，构造出的等价DFA状态数最坏是$O(2^m)$的。其实在很多实际的情况中， NFA或ε-NFA识别输入串的效率也是能被接受的。

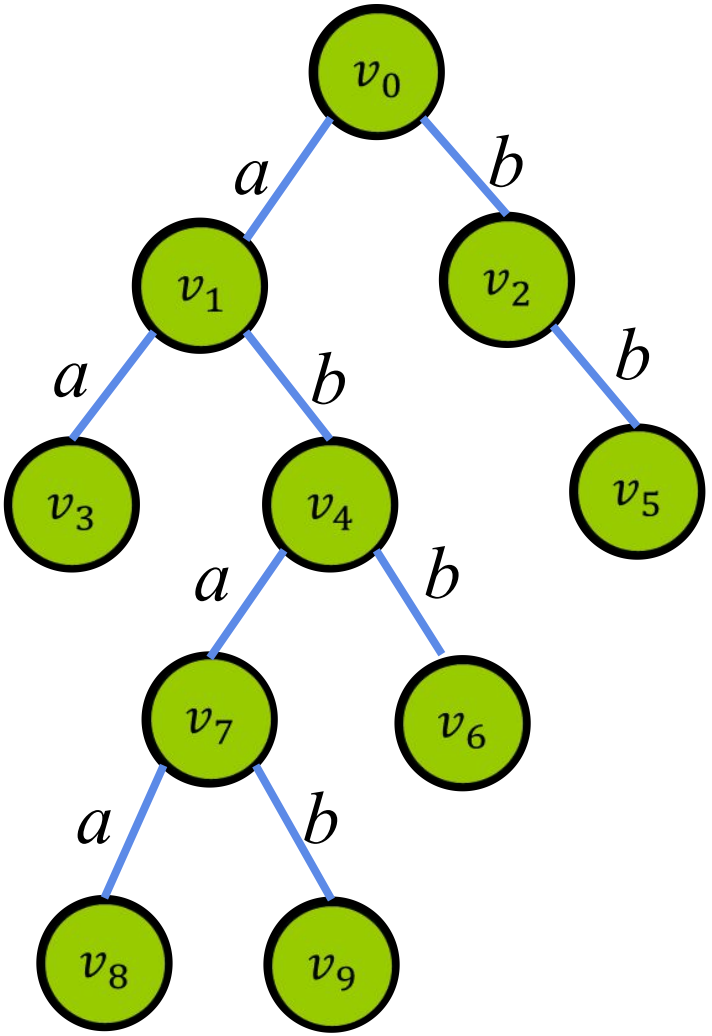
单词查找树

- 下面将要介绍Aho-Corasick自动机，在这之前先引入单词查找树 (Trie)。
- 单词查找树是一棵有根树，每条边上有一个字母表Σ中的字符，并且一个结点向下的边中不存在两条边的字符相同，例如右图就是一棵单词查找树。



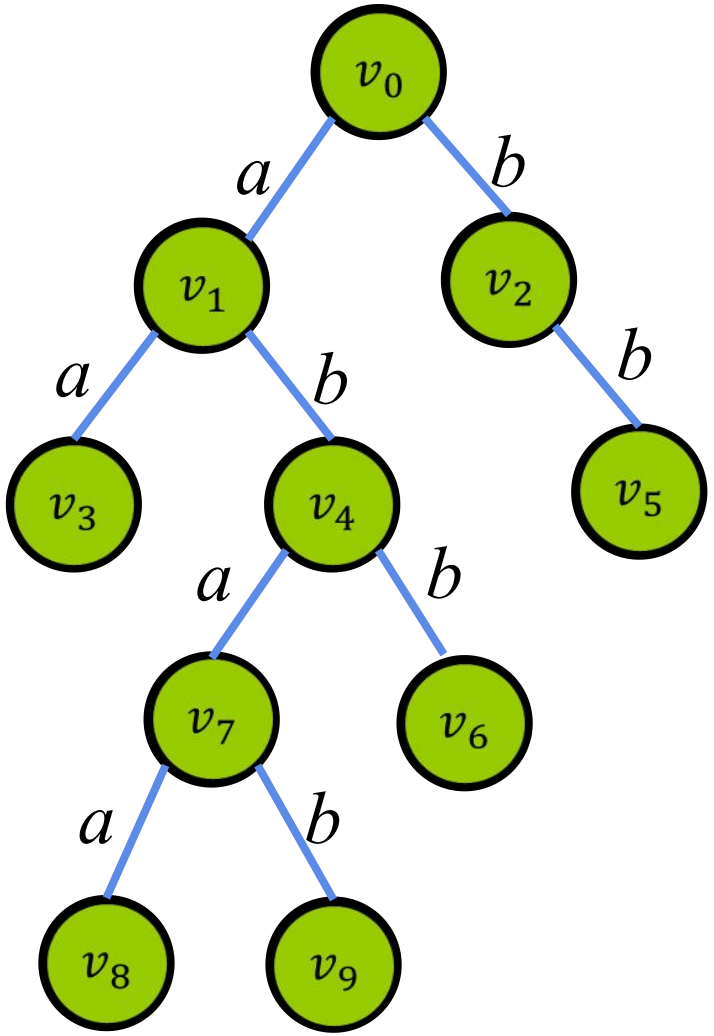
单词查找树

- 在有根树中，根结点到每个结点的路径是唯一的，因此单词查找树中的每一个结点对应了唯一的一个串，也就是从根到它路径上边的字符连成的串。例如结点v_7对应的串是aba。结点v对应的串记做$str(v)$。
- 如果给定字母表Σ上的n个串$w_1, w_2, \ldots w_n$，就能够建出一棵单词查找树，使得对于每个串都存在一个结点和它对应。



单词查找树

- 例如右图的单词查找树，就可以看成五个串*aa*, *abb*, *abaa*, *abab*, *bb*建成的单词查找树。
- 这样实际上就利用了串的公共前缀来表示一个串集合。
- 不难发现建出这棵单词查找树的时间复杂度为$O(|\Sigma|m + \sum_{i=1}^{n}|w_i|)$，其中$m$为单词查找树的结点个数。

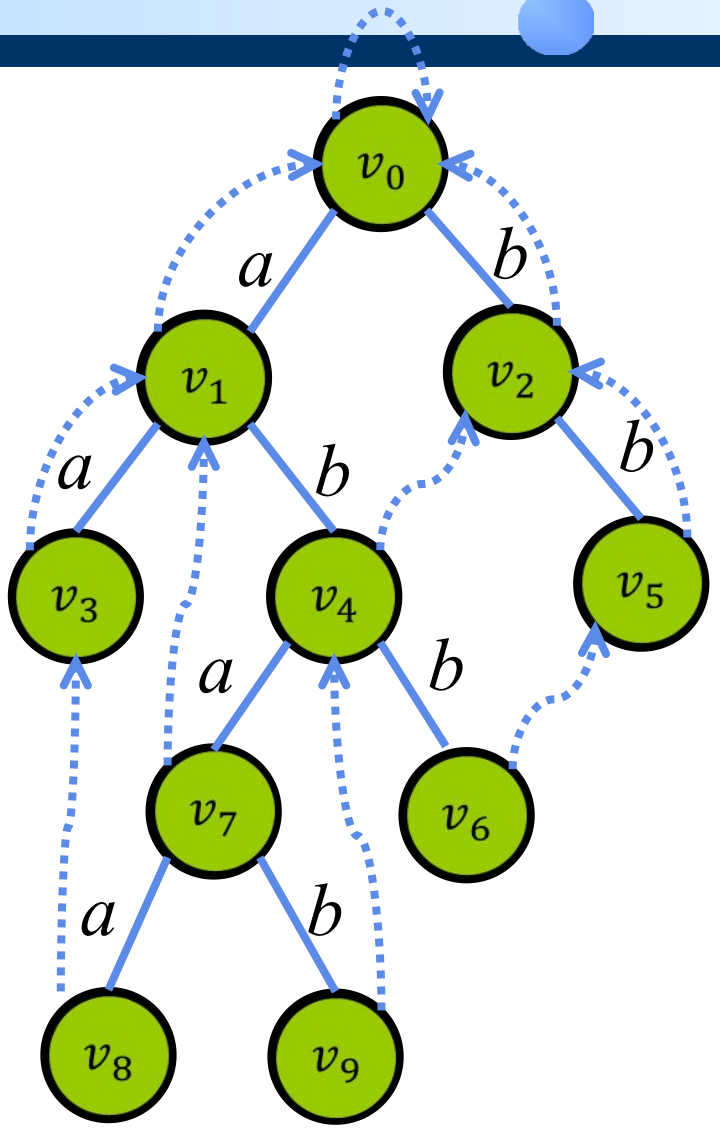


Aho-Corasick自动机

- Aho-Corasick自动机是一个DFA，对于给定字母表Σ上的n个串$w_1, w_2, \ldots w_n$，它能够识别以某个w_i为后缀的串的集合。要构建串$w_1, w_2, \ldots w_n$的Aho-Corasick自动机，先要构建这些串的单词查找树。Aho-Corasick自动机的状态集合就是这棵单词查找树的结点集合，开始状态就是根结点，剩下的问题就是如何构造转移函数和接受状态集合。

前缀指针

- 在构造出最终的Aho-Corasick自动机之前，先定义单词查找树上的前缀指针。
- 单词查找树中，一个结点v的前缀指针$prefix(v)$指向树中另外某个节点，满足$str(prefix(v))$是$str(v)$的某个后缀，同时$prefix(v)$是满足前面条件深度最大的结点。
- 右图用虚线箭头给出了每个结点的前缀指针。



计算前缀指针（算法一）

- 对于结点v，$\left\{str\left(prefix^{k}(v)\right)\right\}$都是$str(v)$的后缀，并且$str(v)$也只有这部分后缀属于这棵单词查找树。
- 由于$str(prefix(v))$是$str(v)$的后缀，因此同时去掉最后一个字符后，前一个串还是后一个串的后缀，这样$prefix(v)$就是$\left\{prefix^{k}\left(parent(v)\right)\right\}$中某个结点的子结点。

计算前缀指针（算法一）

- 下面给出一个计算结点v前缀指针的算法，要求保证深度小于v的结点的前缀指针都已经算出。其中$up(v)$表示结点v向上的边上的字符，$child(v,a)$表示结点v向下且字符为a的边。

procedure GETPREFIXPOINTER(v)

$p \leftarrow prefix(parent(v))$

while $p \neq v_0$ **and** $child(p, up(v)) = null$ **do** $p \leftarrow prefix(p)$

if $parent(p) \neq v_0$ **and** $child(p, up(v)) \neq null$ **then** $p \leftarrow child(p, up(v))$

$prefix(v) \leftarrow p$

end procedure

计算前缀指针（算法一）

- 对于m个结点的单词查找树计算每个结点的前缀指针，前面算法的时间复杂度最坏情况下是$O(m^2)$的。
- 但是如果这棵单词查找树是根据n个串$w_1, w_2, \ldots w_n$构建出的，前面的算法计算前缀指针的总时间复杂度为$O(|\Sigma|m + \sum_{i=1}^{n}|w_i|)$。

计算前缀指针（算法二）

- 下面我们来看另外一个算法，它能在$O(|\Sigma|m)$的时间内构造出所有结点的前缀指针。
- 用$goal(v,a)$表示从结点v开始沿前缀指针往上，第一个不为$null$的$child(p,a)$结点。
- 根据前面的算法容易发现，$prefix(v)$就是$goal(parent(v),up(v))$。同时也能发现，很容易根据已有结果维护$goal(v,a)$。

计算前缀指针（算法二）

- 下面给出新算法的伪代码，要求按照BFS序对每个结点调用下面的过程。

```
procedure GetPrefixPointer(v)
    if v ≠ v_0 and parent(v) ≠ v_0 then
        prefix(v) ← goal(prefix(parent(v)), up(v))
    else
        prefix(v) ← v_0
    end if
    for all a ∈ Σ do
        if child(v, a) ≠ null then
            goal(v, a) ← child(v, a)
        else
            if v ≠ v_0 then goal(v, a) ← goal(prefix(v), a) else goal(v, a) ← v_0
        end if
    end for
end procedure
```

构造完整的DFA

- 最后一步是构造出完整的DFA，也就是构造转移函数δ和接受状态集合F。
- 容易观察出，$\delta(v,a)$其实就是$goal(v,a)$，下面考虑哪些结点属于接受状态。
- 很明显对于每个串w_i，它在单词查找树中对应的结点是接受状态。还有由于$str(prefix(v))$是$str(v)$的后缀，因此如果$prefix(v)$是接受状态，则v也是接受状态，可以按照BFS序构造出完整的接受状态集合。

Aho-Corasick自动机的应用

- Aho-Corasick自动机在多串匹配相关问题中发挥着举足轻重的作用，也是近几年信息学竞赛的一个热门考点，下面通过分析若干例题，来展现它的强大功能。

NOI2011 第一试 阿狸的打字机

- 阿狸有一个栈结构的打字机，支持三种操作：

1. 向栈中压入一个小写英文字母；
2. 弹出栈顶的字母；
3. 将当前栈中的字母从栈底到栈顶依次输出连成一个字符串。

- 给出操作次数不超过$N(N \leq 10^5)$的操作序列，设第i次输出的字符串为w_i，再给出$M(M \leq 10^5)$个询问，每次询问给出两个正整数a, b，要求回答w_a在w_b中出现的次数。

NOI2011 第一试 阿狸的打字机

- 刚看完题会想到把所有输出的字符串弄出来，再考虑用后缀数组之类的工具处理，但是发现输出字符串的总长可能很大，因此要挖掘出输出字符串之间的联系。仔细观察能够发现，阿狸的打字机在操作的过程中，实际上构建出了一棵单词查找树，每个输出的字符串是这棵树中某个结点对应的串。

NOI2011 第一试 阿狸的打字机

- 有了一棵单词查找树，能想到的事就是构建出Aho-Corasick自动机。由于询问的两个字符串都在这棵单词查找树中，因此w_b在DFA中走的路径就是从根结点到w_b对应结点的路径。
- 考虑根结点到w_b对应结点的路径上的每个结点，它们对应的串在w_b中出现次数要加一，并且相应的前缀指针往上走得到的串的出现次数也要加一。注意到前缀指针能够将单词查找树中的结点按照另外一种方式组织成一棵树，计算答案的过程就变成这样： 在原来的单词查找树中从根结点走到w_b对应的结点，每走过一个结点，就在前缀指针树中给相应结点计数值加一，走到w_b后， 就相当于统计w_a对应结点在前缀指针树中，以它为根的子树的计数值的和。

NOI2011 第一试 阿狸的打字机

- 思考到这里，就能想到离线处理所有询问，按照DFS序遍历单词查找树，并且用数据结构维护前缀指针树。能发现最简单的维护方法是用树状数组维护前缀指针树的DFS序。
- 至此问题全部解决了，总时间复杂度为$O(|\Sigma|N+(M+N)\log N)$。

JSOI2009 省选第一试 密码

- 给出$N(N \leq 10)$个长度不超过10的小写英文字母组成的字符串$\{w_i\}$，计算存在多少个由小写英文字母组成且长度为$L(L \leq 25)$的字符串S，满足给出的N个串都作为子串在S中至少出现了一次。
- 如果满足条件的字符串不超过42个，还要输出所有满足条件的字符串。

JSOI2009 省选第一试 密码

- 首先把给出的N个串建成Aho-Corasick自动机，这里为了能在后面判断是否包含了某个w_i，需要对每个接受状态记录哪些w_i被作为后缀识别出来了，在构建Aho-Corasick自动机的最后一步可以通过前缀指针求出。 由于$N \leq 10$，可以用二进制状态表示一个集合。
- 然后考虑用动态规划算法计算答案，用状态$f(i,q,S)$表示长度为i的字符串、在DFA中走到了状态q、并且已经被识别出的串集合为S时的方案数。通过枚举串的下一个字符，可以$O(1)$转移。整个动态规划算法的时间复杂度为$O(|\Sigma|LM2^N)$，其中M表示DFA的状态数。

JSOI2009 省选第一试 密码

- 最后一个的问题是如何输出所有满足条件的字符串，由于满足条件的字符串个数不超过42，容易发现不存在某个字符是可以去掉的（否则方案数至少为52）。这样的话，可以用搜索的方法找到所有满足条件的字符串，一个很有效的搜索方案是每次选择某个还没出现过的w_i放在当前S的最后（可能和原先的串有重叠部分）。至此问题全部解决。

JSOI2009 差额第二试 有趣的游戏

- 有$N(N \leq 10)$个人在玩一个游戏，每个人有一个由前$M(M \leq 10)$个大写英文组成的字符串，每个人的字符串不相同并且长度都为$L(L \leq 10)$。有一个机器每次随机产生一个字符，产生的字符被依次连成一个字符串，如果某个人发现自己的字符串在机器生成的字符串中出现了，游戏结束并且这个人就赢了。 给出前M个大写字母被生成的概率（保证和正好为1），计算每个人赢的概率。

JSOI2009 差额第二试 有趣的游戏

- 首先考虑判断没有人能赢的特殊情况，如果出现这种情况，那么每个人的字符串中都存在某个字符的产生概率为0，否则的话，所有人赢的概率和一定为1。
- 将给出的N个串建成Aho-Corasick自动机，用$p(w)$表示串w产生的概率、$v(w)$表示串w最终走到的状态，注意如果走到接受状态的话，就一定会停止，因此会出现某些串w满足$p(w) = 0$且$v(w) = null$，为了后面方便，将接受状态的转移都去掉。
- 令$x(q) = \sum_{v(w)=q} p(w)$，容易发现，如果第$i$个人的字符串对应的接受状态是$q$，则$x(q)$就是他赢的概率。

JSOI2009 差额第二试 有趣的游戏

- 考虑某个状态q，对于所有存在的转移$\delta(p_i, a_i) = q$，能发现满足$x(q) = \sum p(a_i)x(p_i)$，这样就能够列出线性方程组。尝试解出$x(q)$时会发现，$x(q)$全为0一定是一个解，这说明系数矩阵中存在线性相关的行，随意去掉一行后，加入方程$\sum_{q \in F} x(q) = 1$，这样就能顺利解出所有$x(q)$。

总结

- 字符串处理是信息学竞赛中的一个重要考点，有限状态自动机更是在字符串匹配问题中发挥了重要的作用。从近几年的竞赛题看，Aho-Corasick自动机相关题目出现了不少，后缀自动机也能在一定程度上替代后缀树、后缀数组等传统后缀结构，这样更让我们看到了有限状态机在信息学竞赛中的巨大价值。
- 信息学竞赛中的学习是一个思考的过程，更是一个总结的过程。本文只是作为一个抛砖引玉的作用，提出一些思考问题的方法，对我自己思考过的问题进行总结，希望借此读者能够激发出更多的灵感，发现有限状态自动机更多美妙的性质，并将它们用到实际的解题过程中。

提问时间

JSOI2012 省选第一试 玄武密码

- 给出一个长度为$N(N \leq 10^7)$的字符串S，S的每个字符是'E'、'S'、'W'、'E'中的一个。再给出$M(M \leq 10^5)$个长度不超过100的字符串$\{w_i\}$，这些字符串的每个字符也是'E'、'S'、'W'、'E'中的一个。要求对于这M个字符串中的每一个w_i，计算它在S中出现的最长前缀长度。

JSOI2012 省选第一试 玄武密码

- 对给出的M个字符串构建Aho-Corasick自动机，然后将S在DFA里走一遍，标记到达过的状态。对于被标记过的状态，说明它在单词查找树中对应的串在S中作为子串出现过。同时注意到，如果单词查找树中的某个结点对应的串出现在了S中，它前缀指针指向结点对应的串也出现在S中。通过逆BFS序，可以将相应的前缀指针结点都标记上。标记完成后可以发现，题目要求计算的东西，实际上就是每个字符串对应结点往上遇到的第一个有标记结点的深度。BFS一遍单词查找树就能得到所有M个串的答案。
- 算法的时间复杂度为$O(|\Sigma|T+\sum_{i=1}^{M}|w_i|+|S|)$，空间复杂度为$O(|\Sigma|T)$，其中$T$表示AC自动机的结点数。

CTSC2010 第二试 珠宝商

- 给出一棵N个结点的树，每个结点上有一个小写字母，用$S_{u\to v}$表示结点u到结点v路径经过的结点（包括u和v）上的字符组成的字符串，一般来说，$S_{u\to v}$和$S_{v\to u}$是不同的。再给出一个长度为M的字符串P，用$C(S)$表示字符串S在P中出现的次数（也就是作为子串的次数），求$\sum_{u,v\in V} C(S_{u\to v})$。
- 对于40%的数据，$N \le 8{,}000,\ M \le 50{,}000$；对于100%的数据，$N, M \le 50{,}000$。时间限制为每个测试点18秒。

CTSC2010 第二试 珠宝商

- 考虑枚举每个点r作为树根，计算$\sum_{i\in v} C(S_{r\to i})$。由于本身就是树结构，所以很容易在$O(N)$的时间内将$\{S_{r\to i}\}$建成一棵单词查找树。然后通过构造前缀指针使其变为Aho-Corasick自动机，需要注意这里是给出了单词查找树，要使用前面的第二种算法构造前缀指针。
- 下面将串P在DFA里走一遍，如果某次走到状态q，说明单词查找树中相应结点v对应的串在P中出现了一次，并且$\left\{str\left(prefix^k(v)\right)\right\}$在$P$中的出现次数也都要加一。这里暂时只将$v$的计数值加一，最后通过逆BFS序可以统计每个结点实际的计数值。
- 这个算法的时间复杂度为$O(|\Sigma|N^2+NM)$，能够在规定时限内通过40%的数据。